**Java高并发秒杀API之业务分析**

**文档修改历史**

| 版本号 | 修改日期 | 编写 | 评审 | 批准 | 修改内容 |  |
| --- | --- | --- | --- | --- | --- | --- |
| V0.1 | 2016-10-24 |  |  |  | 新建 |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

**资邦金服网络科技集团有限公司**

**2016年10月**

目录

[1 Java高并发秒杀API之业务分析 3](#_Toc468895029)

[1.1 第1章 课程介绍 3](#_Toc468895030)

[1.1.1 课程介绍 3](#_Toc468895031)

[1.1.2 项目效果演示 4](#_Toc468895032)

[1.2 第2章 梳理所有技术和搭建工程 5](#_Toc468895033)

[1.2.1 相关技术介绍 5](#_Toc468895034)

[1.2.2 创建项目和依赖 5](#_Toc468895035)

[1.3 第3章 秒杀业务分析 7](#_Toc468895036)

[1.3.1 秒杀业务分析 7](#_Toc468895037)

[1.3.2 MySQL实现秒杀难点分析 9](#_Toc468895038)

[1.3.3 实现哪些秒杀功能 10](#_Toc468895039)

[1.4 第4章 DAO层设计与开发 11](#_Toc468895040)

[1.4.1 数据库设计与编码 12](#_Toc468895041)

[1.4.2 DAO实体和接口编码 13](#_Toc468895042)

[1.4.3 基于myBatis实现DAO理论 14](#_Toc468895043)

[1.4.4 基于myBatis实现DAO编程(上) 16](#_Toc468895044)

[1.4.5 基于myBatis实现DAO编程(下) 16](#_Toc468895045)

[1.4.6 myBatis整合Spring理论 17](#_Toc468895046)

[1.4.7 mybatis整合Spring编码 19](#_Toc468895047)

[1.4.8 DAO层单元测试编码和问题排查(上) 21](#_Toc468895048)

[1.4.9 DAO层单元测试编码和问题排查(下) 22](#_Toc468895049)

# Java高并发秒杀API之业务分析

## ****第1章 课程介绍****

本章介绍秒杀系统的技术内容，以及系统演示。并介绍不同程度的学员可以学到什么内容。

### 课程介绍

pringMVC+Spring+MyBatis使用与整合秒杀类系统需求理解和实现常用技术解决高并发问题（java web、前端、MySQL）

为什么使用这三个框架？   
互联网公司常用框架（阿里、京东、搜狐、美团…）   
框架易于使用和轻量级   
底代码侵入性   
成熟的社区和用户群

为什么用秒杀系统来讲本课程？   
秒杀业务场景具有典型“事务”特性   
秒杀/红包类需求越来越常见   
面试常问问题（如何设计一个秒杀系统和优化一个秒杀系统）

从本课程学到什么？   
初学者：框架的使用与整合 技巧   
有经验者：秒杀分析过程和优化思路

秒杀系列将分为四门课程进行，分别是：   
Java高并发秒杀API之业务分析与DAO层   
Java高并发秒杀API之Service   
Java高并发秒杀API之web   
Java高并发秒杀API之高并发优化

### 项目效果演示













## ****第2章 梳理所有技术和搭建工程****

本章首先介绍秒杀系统所用框架和技术点，然后介绍如何基于maven搭建项目，最后对工程目录包进行了划分。

### 相关技术介绍

用到那些技术？   
MySQL（表设计 、SQL技巧、事务和行级锁）   
MyBatis（DAO层设计与开发、MyBatis合理使用、MyBatis与Spring整合）   
Spring（Spring IOC整合Service、声明式事务运用）   
SpringMVC（Restful接口设计和使用、框架运作流程、controller开发技巧）   
前端（交互设计、Bootstrap、jQuery）   
高并发（高并发点和高并发分析、优化思路并实现）

基于maven创建项目

### 创建项目和依赖

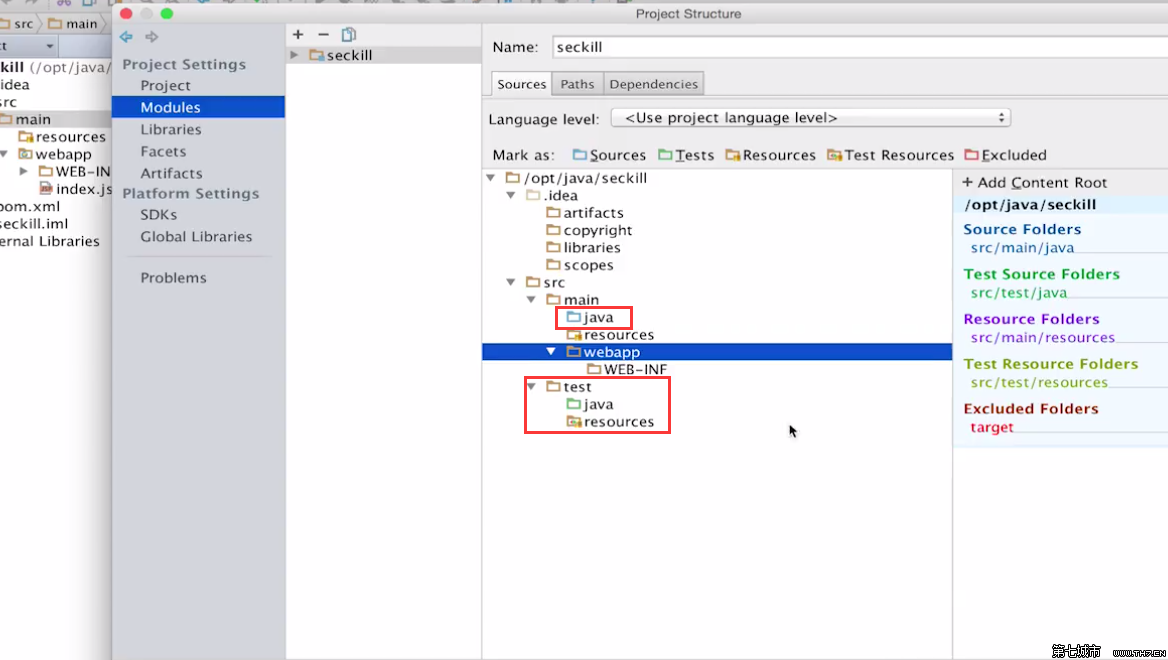
开始创建项目之前的说明   
从零开始创建   
从官网获取相关配置   
使用Maven创建项目

为什么从官网获取资源   
文档更全面权威   
避免过时货错误   
官网地址：   
logback配置：http://logback.qos.ch/manual/configuration.html   
spring配置：http://docs.spring.io/spring/docs/   
mybatis配置：http://mybatis.github.io/mybatis-3/zh/index.html

开始创建项目   
maven命令创建web骨架项目   
**mvn archetype:generate -DarchetypeCatalog=internal -DgroupId=com.seckill -DartifactId=seckill -DarchetypeArtifactId=maven-archetype-webapp**   
注：-DgroupId和-DartifactId标注项目的坐标，项目叫com.seckill，-DarchetypeArtifactId=maven-archetype-webapp表示使用maven的原型webapp的原型去创建项目

下面就通过IntelliJ IDEA或Eclipse开发工具Import项目

**存在的问题：**   
1.用maven创建的webapp要修改servlet的版本：   
maven 创建项目的web.xml 版本比较低（Servlet2.3，jsp默认的el表达式是不工作的），所以需要把他切换到更高的Servlet版本，可以去tomcat webapps包下面去找到示例项目的web.xml的头copy过来。如果使用的IntelliJ IDEA导入的项目需要把一些没有的目录补全，如下示例（仅供参考）：

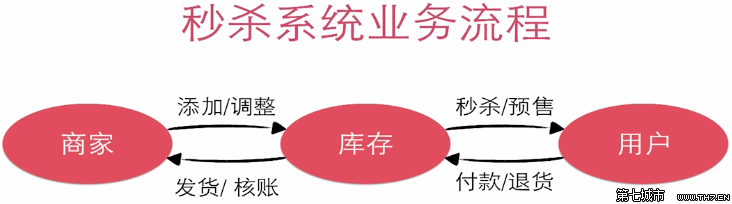


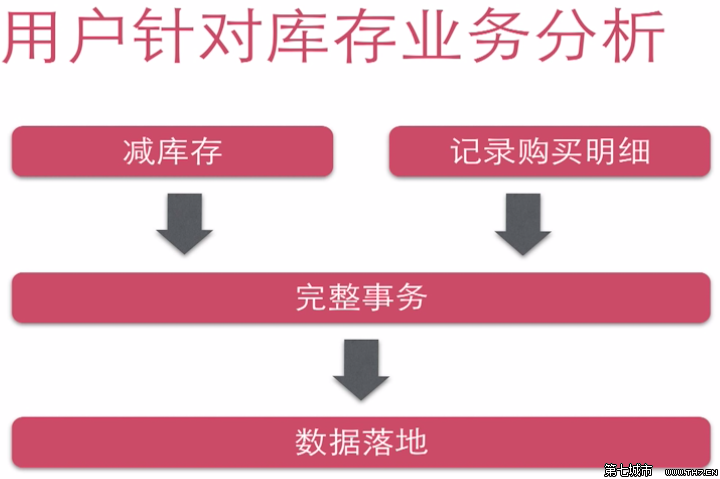
打开maven的pom.xml文件把默认的junit版本改为4.11是应为3.0的junit默认使用编程的方式，4.0是使用注解的方式来运行junit。再接下来就是补全项目的依赖

## ****第3章 秒杀业务分析****

本章讲解常见秒杀业务以及如何用最常用的技术实现。分析了秒杀业务的难点，以及本课程要实现哪些秒杀API。

### 秒杀业务分析







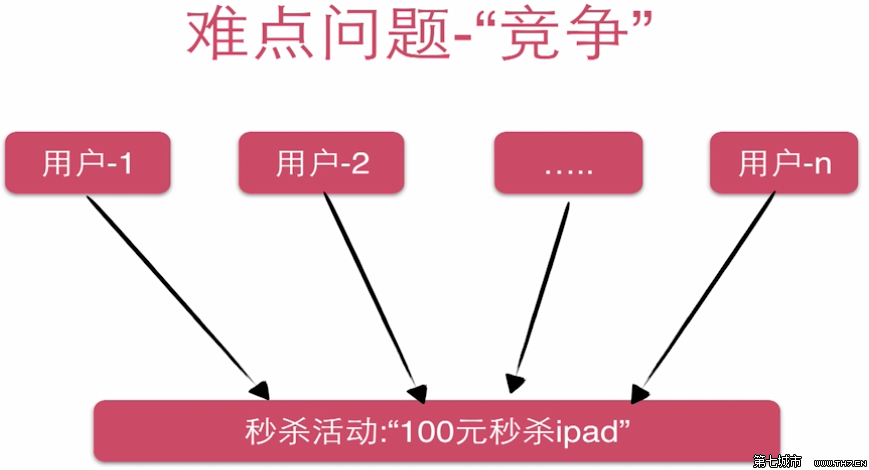
**为什么需要事物？**





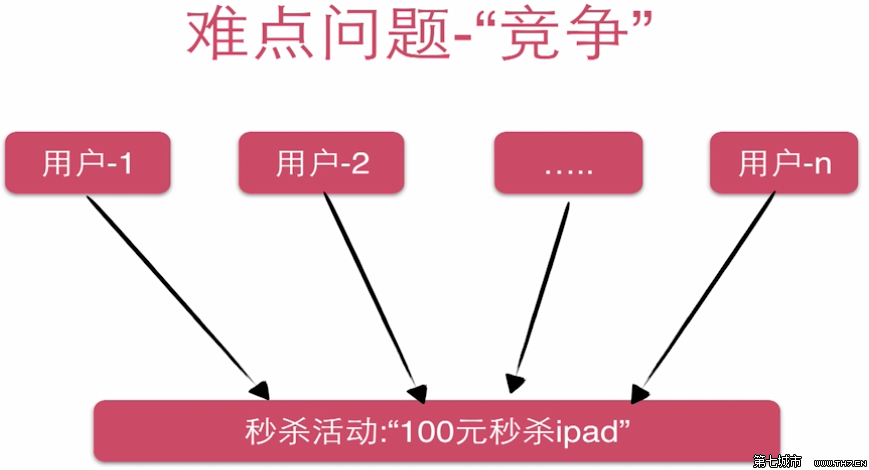
关于数据落地：MySQL VS NoSQL（事务依然是目前最可靠的落地方案，NoSQL对事物的支持做的不是很好。）

### MySQL实现秒杀难点分析



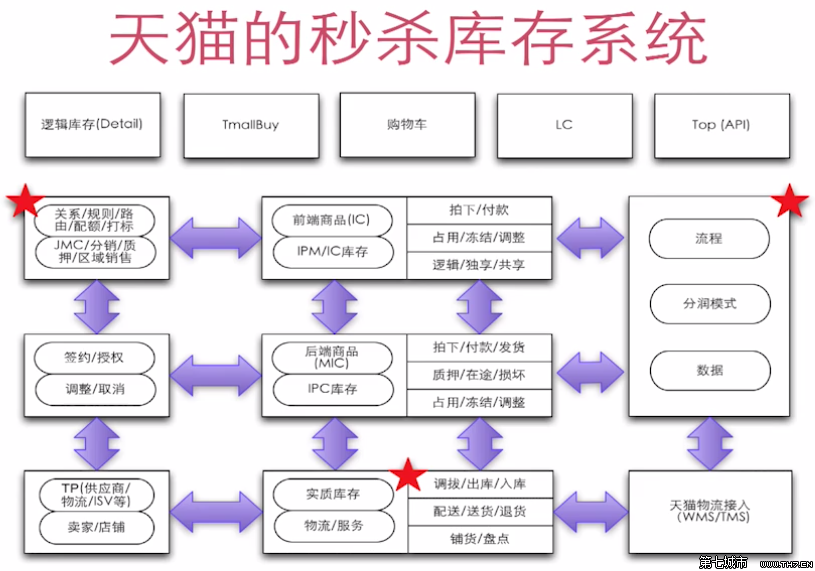
对于MySQL来说竞争反应到背后的技术是怎样的呢?(事物+行级锁)

**事物：**   
Start Transaction   
**Update库存数量**   
Insert购买明细   
Commit



秒杀的难点是如何高效的处理竞争？   
知道问题后如何解决？

### 实现哪些秒杀功能







## ****第4章 DAO层设计与开发****

本章介绍秒杀系统数据库设计与实现，分析DAO数据持久化层所需接口，并编码实现。以及MyBatis如何与spring进行整合，最后介绍如何测试整合框架，完成DAO层的单元测试。

### 数据库设计与编码

--数据库初始化

--创建数据库CREATE DATABASE seckill;

--使用数据库use seckill;

--创建秒杀库存表

CREATE table seckill(

`seckill\_id` bigint NOT NULL AUTO\_INCREMENT COMMENT '商品库存id', `name` varchar(120) NOT NULL COMMENT '商品名称',

`number` int NOT NULL COMMENT '库存数量',

`start\_time` timestamp NOT NULL COMMENT '秒杀开启时间',

`end\_time` timestamp NOT NULL COMMENT '秒杀结束时间',

`create\_time` timestamp NOT NULL DEFAULT CURRENT\_TIMESTAMP COMMENT '创建时间',

PRIMARY KEY (seckill\_id),

key idx\_start\_time(start\_time),

key idx\_end\_time(end\_time),

key idx\_create\_time(create\_time))

ENGINE=InnoDB AUTO\_INCREMENT=1000 DEFAULT CHARSET=utf8 COMMENT='描述库存表';

--创建索引，用于加速查询

--key idx\_start\_time(start\_time),

--key idx\_end\_time(end\_time),

--key idx\_create\_time(create\_time)

--默认mysql有多种存储引擎，但是可以支持事物的只有InnoDB,所以在这里通过显示的语法去告诉mysql这个表的ENGINE引擎是InnoDB

--由于数据库采用的是自增作为主键，给出初始的自增id:AUTO\_INCREMENT=1000

--默认编码 DEFAULT CHARSET=utf-8

--表的注释 COMMENT='秒杀库存表'

--初始化数据

insert into seckill(name,number,start\_time,end\_time)value

('1000元秒杀iphone6',100,'2012-12-12 00:00:00','2012-12-13 00:00:00'), ('500元秒杀ipad2',200,'2012-12-12 00:00:00','2012-12-13 00:00:00'), ('300元秒杀mi4',300,'2012-12-12 00:00:00','2012-12-13 00:00:00'), ('200元秒杀红米note',400,'2012-12-12 00:00:00','2012-12-13 00:00:00');

--秒杀成功明细

--用户登录认证相关的信息

create table success\_killed( `seckill\_id` bigint not null COMMENT '秒杀商品id', `user\_phone` bigint not null comment '用户手机号', `state` tinyint not null default -1 comment '状态标识：-1：无效 0：成功 1：已付款 2：已发货', `create\_time` timestamp not null comment '创建时间', primary key(seckill\_id,user\_phone),/\*联合主键\*/ key idx\_create\_time(create\_time))ENGINE=InnoDB DEFAULT CHARSET=utf8 COMMENT='秒杀成功明细表';

--连接数据库控制台

mysql -uroot -p123456

--查看创建表的结构

show create table seckill/G

--为什么手写DDL

--记录每次上线的DDL修改

--上线V1.1

ALTER TABLE seckillDROP INDEX idx\_create\_time,ADD INDEX idx\_c\_s(start\_time,create\_time);

--上线V1.2

--DDL

### DAO实体和接口编码

Table —-> Entity

package org.seckill.entity;import java.util.Date;

/\*\* \* 秒杀库存实体 \* @author Administrator \* \*/

public class Seckill {

private long seckillId;

private String name;

private int number;

private Date startTime;

private Date endTime;

private Date createTime;

public long getSeckillId() { return seckillId; }

public void setSeckillId(long seckillId) {

this.seckillId = seckillId;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public int getNumber() {

return number;

}

public void setNumber(int number) {

this.number = number;

}

public Date getStartTime() {

return startTime;

}

public void setStartTime(Date startTime) {

this.startTime = startTime;

}

public Date getEndTime() {

return endTime;

}

public void setEndTime(Date endTime) {

this.endTime = endTime;

}

public Date getCreateTime() {

return createTime;

}

public void setCreateTime(Date createTime) {

this.createTime = createTime;

}

@Override

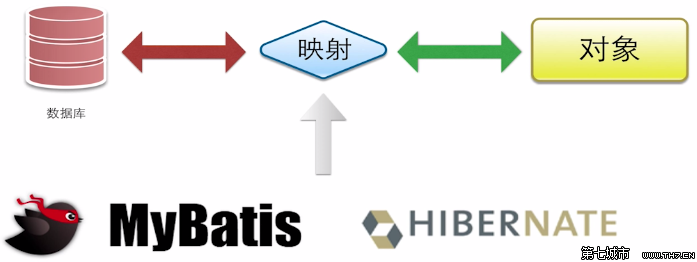
public String toString() {

return "Seckill{" + "seckillId=" + seckillId + ", name='" + name + '/'' + ", number=" + number + ", startTime=" + startTime + ", endTime=" + endTime + ", createTime=" + createTime + '}'; }

}

### 基于myBatis实现DAO理论

MyBatis用来做什么？



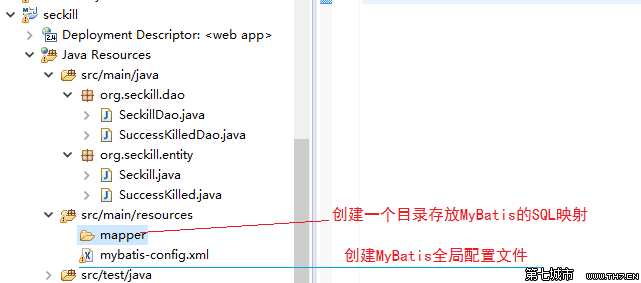


MyBatis怎么用   
SQL写在哪？（**XML提供SQL**、注解提供SQL，注：注解是java5.0之后提供的一个新特性）   
对于实际的使用中建议使用XML文件的方式提供SQL，通过注解的方式呢，注解本身还是java源码，修改和调整SQL其实是非常不方便的，一样需要重新编译类，当我们写复杂的SQL尤其拼接逻辑时，注解处理起来就会非常繁琐，那么XML提供了很多的SQL拼接和处理逻辑的标签可以非常方便的帮我们去做封装。

如何DAO接口接口   
Mapper自动实现DAO（也就是DAO只需要设计接口，不需要去写实现类，MyBatis知道我们的参数、返回类型是什么同时也有SQL文件，他可以制动帮我们生成接口的实现类来帮我们执行参数的封装，执行SQL，把我们的返回结果集封装成我们想要的类型）   
第二种是通过API编程方式实现DAO接口（MyBatis通过给我们提供了非常多的API，跟其他的ORM和JDBC很像）

在实际开发中建议使用Mapper自动实现DAO，这样可以直接只关注SQL如何编写，如何去设计DAO接口，帮我们节省了很多的维护程序，所有的实现都是MyBatis自动完成。

### 基于myBatis实现DAO编程(上)



打开MyBatis的官方文档   
http://www.mybatis.org/mybatis-3/zh/index.html   
入门——>找到MyBatis全局配置，里面有XML的规范（XML的标签约束dtd文件）拷入到项目的MyBatis全局配置文件中，开始配置MyBatis，如下：

<?xml version="1.0" encoding="UTF-8" ?><!DOCTYPE configuration PUBLIC "-//mybatis.org//DTD Config 3.0//EN" "http://mybatis.org/dtd/mybatis-3-config.dtd"><configuration> <!-- 配置全局属性 --> <settings> <!--使用jdbc的getGeneratekeys获取自增主键值，当inert一条记录时我们是不插入id的，id是通过自增去赋值的，当插入完后想得到该插入记录的id时可以调用jdbc的getGeneratekeys --> <setting name="useGeneratedKeys" value="true" /> <!--使用列别名替换别名 默认true select name as title form table; --> <setting name="useColumnLabel" value="true" /> <!--开启驼峰命名转换Table:create\_time到 Entity(createTime) --> <setting name="mapUnderscoreToCamelCase" value="true" /> </settings></configuration>

使用Mapper自动实现DAO   
在mapper目录中创建SeckillDao.xml和SuccessKilledDao.xml

<?xml version="1.0" encoding="UTF-8" ?><!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://mybatis.org/dtd/myba

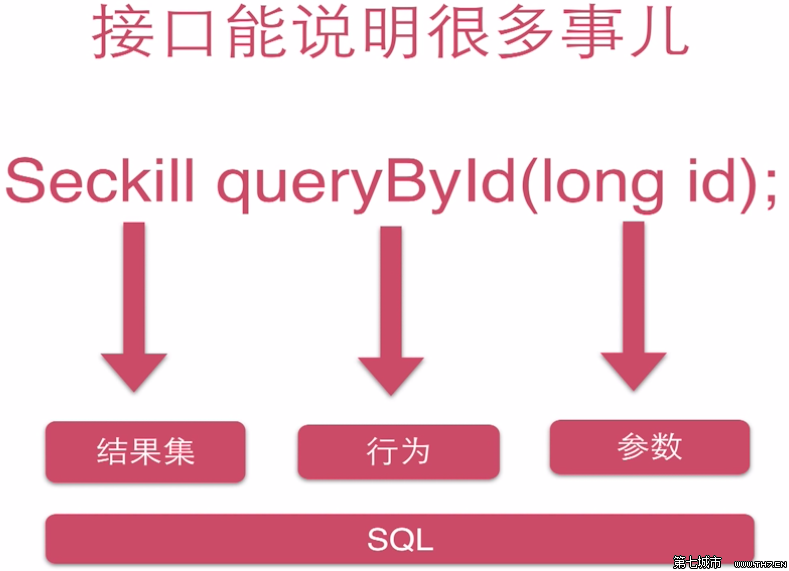
### 基于myBatis实现DAO编程(下)

<?xml version="1.0" encoding="UTF-8" ?><!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://mybatis.org/dtd/mybatis-3-mapper.dtd"><mapper namespace="org.seckill.dao.SuccessKilledDao"> <!--通过ignore关键字将主键冲突时的报错改为返回0--> <!--int insertSuccessKilled(long seckilledId,long userPhone);--> <update id="insertSuccessKilled"> INSERT ignore INTO success\_killed(seckill\_id,user\_phone,state)VALUES (#{seckilledId},#{userPhone},1) </update> <!-- SuccessKilled queryByIdWithSeckill(@Param("seckilledId") long seckilledId, @Param("userPhone") long userPhone);--> <select id="queryByIdWithSeckill" resultType="SuccessKilled"> SELECT sk.seckill\_id,sk.user\_phone,sk.create\_time,sk.state, s.seckill\_id "seckill.seckill\_id",s.name "seckill.name", s.start\_time "seckill.start\_time",s.end\_time "seckill.end\_time", s.create\_time "seckill.create\_time" FROM success\_killed sk INNER JOIN seckill s ON sk.seckill\_id=s.seckill\_id WHERE sk.seckill\_id=#{seckilledId} and sk.user\_phone=#{userPhone}; </select></mapper>

注：上面的s.seckill\_id “seckill.seckill\_id”表示将s.seckill\_id这一列的数据是seckill属性里的seckill\_id属性，是一个级联的过程，使用的就是别名只是忽略了as关键字，别名要加上双引号。

### myBatis整合Spring理论

整合目标   
更少的编码（只写接口，不写实现，MyBatis实现DAO）   
更少的配置   
足够的灵活性











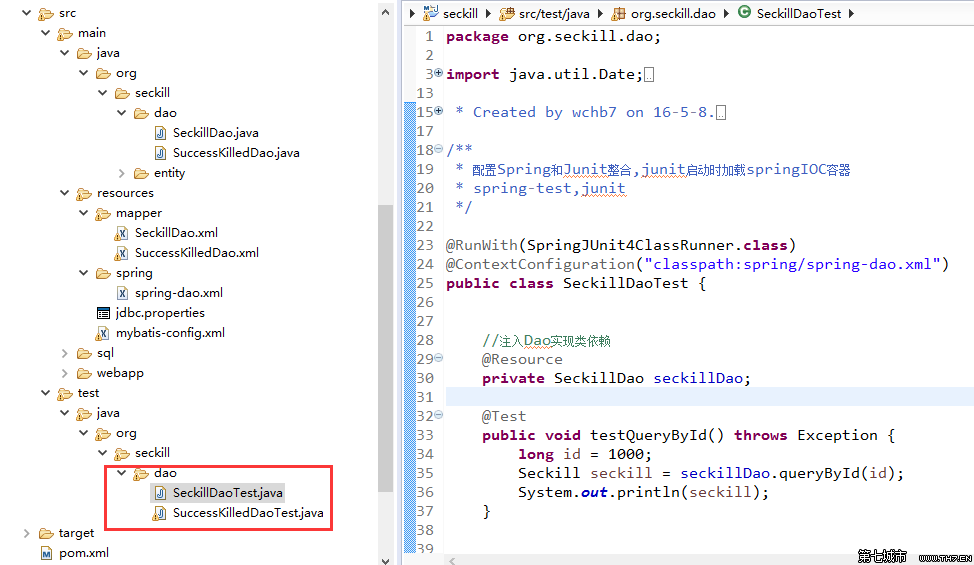
XML提供SQL，DAO接口Mapper来作为接口实现类时完全可以依赖于Spring去帮我们做好这些事情，当用Spring整合MyBatis时建议使用这两种方式ML提供SQL，DAO接口提供Mapper。   


### mybatis整合Spring编码

在resource目录下创建一个新的目录Spring(存放所有spring相关的配置)

<?xml version="1.0" encoding="UTF-8"?><beans xmlns="http://www.springframework.org/schema/beans" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:context="http://www.springframework.org/schema/context" xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context.xsd"> <!--配置整合mybatis过程--> <!--１.配置数据库相关参数--> <context:property-placeholder location="classpath:jdbc.properties"/> <!--2.数据库连接池--> <!--todo java.sql.SQLException: Access denied for user ''@'localhost' (using password: NO)--> <bean id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource"> <!-- 配置连接池属性 --> <property name="driverClass" value="${driverClassName}"></property> <property name="jdbcUrl" value="${jdbc.url}"></property> <property name="user" value="${jdbc.username}"></property> <property name="password" value="${jdbc.password}"></property><!-- <property value="com.mysql.jdbc.Driver"></property> <property value="jdbc:mysql://localhost:3306/test\_mysql"></property> <property value="jdbc:mysql://127.0.0.1:3306/test\_mysql"></property> <property value="root"></property> <property value="111111"></property> --> <!-- c3p0连接池的私有属性 --> <property name="maxPoolSize" value="30"></property> <property name="minPoolSize" value="10"></property> <!-- 关闭连接后不自动commit --> <property name="autoCommitOnClose" value="false"></property> <!-- 获取连接超时时间 --> <property name="checkoutTimeout" value="10000"></property> <!-- 获取连接重试次数 --> <property name="acquireRetryAttempts" value="3"></property> </bean><!-- <bean id="dataSource"> <property value="com.mysql.jdbc.Driver"/> <property value="jdbc:mysql://localhost:3306/test\_mysql"/> <property value="root"/> <property value="111111"/> todo <property value="${driverClassName}"/> <property value="${jdbc.url}"/> <property value="${jdbc.username}"/> <property value="${jdbc.password}"/> </bean> --> <!--３.配置SqlSessionFactory对象--> <bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean"> <!--注入数据库连接池--> <property name="dataSource" ref="dataSource"/> <!--配置mybatis全局配置文件:mybatis-config.xml--> <property name="configLocation" value="classpath:mybatis-config.xml"/> <!--扫描entity包,使用别名,多个用;隔开--> <property name="typeAliasesPackage" value="org.seckill.entity"/> <!--扫描sql配置文件:mapper需要的xml文件--> <property name="mapperLocations" value="classpath:mapper/\*.xml"/> </bean> <!--４:配置扫描Dao接口包,动态实现DAO接口,注入到spring容器--> <bean class="org.mybatis.spring.mapper.MapperScannerConfigurer"> <!--注入SqlSessionFactory--> <property name="sqlSessionFactoryBeanName" value="sqlSessionFactory"/> <!-- 给出需要扫描的Dao接口--> <property name="basePackage" value="org.seckill.dao"/> </bean> <!--RedisDao--> <!-- <bean id="redisDao"> <constructor-arg index="0" value="localhost"/> <constructor-arg index="1" value="6379"/> </bean> --></beans>

### DAO层单元测试编码和问题排查(上)



package org.seckill.dao;import java.util.Date;import java.util.List;import javax.annotation.Resource;import org.junit.Test;import org.junit.runner.RunWith;import org.seckill.entity.Seckill;import org.springframework.test.context.ContextConfiguration;import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;/\*\* \* Created by wchb7 on 16-5-8. \*//\*\* \* 配置Spring和Junit整合,junit启动时加载springIOC容器 \* spring-test,junit \*/@RunWith(SpringJUnit4ClassRunner.class)@ContextConfiguration("classpath:spring/spring-dao.xml")public class SeckillDaoTest { //注入Dao实现类依赖 @Resource private SeckillDao seckillDao; @Test public void testQueryById() throws Exception { long id = 1000; Seckill seckill = seckillDao.queryById(id); System.out.println(seckill); } @Test public void testQueryAll() throws Exception {// Java没有保存形参的记录:QueryAll(int offset,int limit)->QueryAll(arg0,arg1);// 因为java形参的问题,多个基本类型参数的时候需要用@Param("seckillId")注解区分开来 List<Seckill> seckills = seckillDao.queryAll(0, 4); for (Seckill seckill : seckills) { System.out.println(seckill); } } @Test public void testReduceNumber() throws Exception { Date killTime = new Date(); int updateCount = seckillDao.reduceNumber(1000L, killTime); System.out.println("updateCount: " + updateCount); }}

### DAO层单元测试编码和问题排查(下)

package org.seckill.dao;import org.junit.Test;import org.junit.runner.RunWith;import org.seckill.entity.SuccessKilled;import org.springframework.test.context.ContextConfiguration;import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;import javax.annotation.Resource;import static org.junit.Assert.\*;/\*\* \* Created by wchb7 on 16-5-9. \*/@RunWith(SpringJUnit4ClassRunner.class)@ContextConfiguration("classpath:spring/spring-dao.xml")public class SuccessKilledDaoTest { @Resource private SuccessKilledDao successKilledDao; @Test public void testInsertSuccessKilled() throws Exception { /\* 第一次：iinsertCount=1 第一次：iinsertCount=0 \*/ long id = 1000L; long phone = 15811112222L; int insertCount = successKilledDao.insertSuccessKilled(id, phone); System.out.println("insertCount: " + insertCount); } @Test public void testQueryByIdWithSeckill() throws Exception { long id = 1000L; long phone = 15811112222L; SuccessKilled successKilled = successKilledDao.queryByIdWithSeckill(id, phone); System.out.println(successKilled); if (successKilled != null) { System.out.println(successKilled.getSeckill()); } }}